SimpleModelingとSimpleModeler

モデル駆動開発をターゲットにしたオブジェクト・モデリング手法とScala DSLによるモデル・コンパイラ

浅海智晴

2008年11月14日

浅海のプロフィール

- 1985年-2001年：富士通
  - UNIX OSをビジネス向けに改造する仕事
    - ファイル管理、分散ファイルシステム、Webサーバなど
    - 信頼性、運用管理、COBOL向けの改造
  - 1993年頃からオブジェクト・モデリングの調査を始める
  - 1995年からJavaの利用を始める
  - 1998年からJava&XMLのフリーソフトを開発・公開(個人活動)
    - SmartDoc(XML文書処理系)、Relaxer(プログラム自動生成)
- 2001年-現在：浅海智晴事務所代表
  - モデリング、XML、Javaのコンサルティング、教育活動
- 2002、2003年度：IPA未踏に採用
  - Relaxer (DSLによるプログラムの自動生成)
- 2005年度-2007年度：稚内北星学園大学東京サテライト校教授
- 2007年度-現在：日本Javaユーザグループ副会長

SimpleModelingの本

開発プログラム

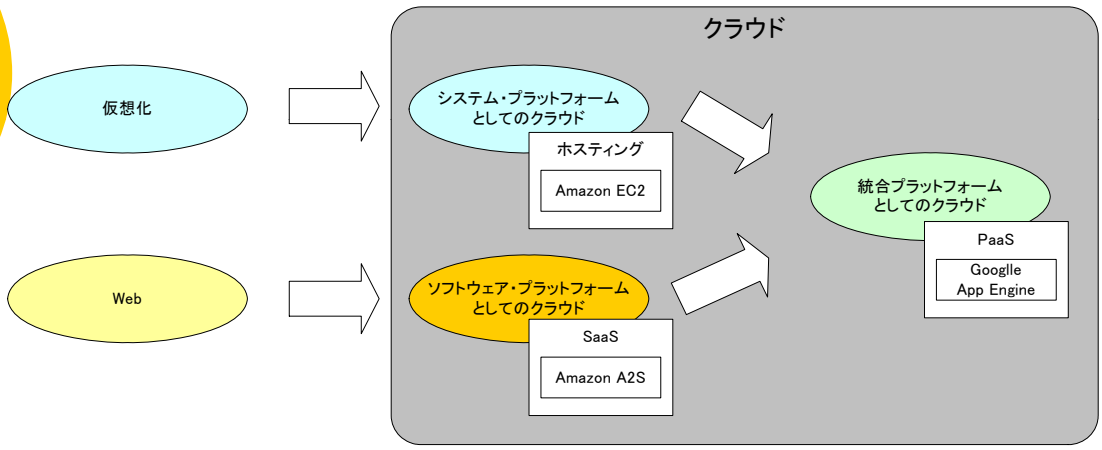
- SmartDoc (1998年)
  - XML文書処理系
  - 専用XML文書からHTML、LaTeX、プレインテキストを生成
- Relaxer (2000年)
  - XMLスキーマ言語RELAXをDSLとして用いたスキーマ・コンパイラ
  - RELAXからJavaプログラム、W3C XML Schemaなどを生成
- SmartCase (2004年、試作)
  - 専用XML文書でユースケース・モデルを記述
  - 仕様書を生成
- JavaDSL (2007年、試作)
  - JavaをDSLのメタ言語としてオブジェクト・モデルを記述
  - Javaプログラムと仕様書を生成

テーマ

- プログラマのためのモデリング
  - モデリング手法とモデル駆動開発
- モデリング手法
  - SimpleModeling
- モデル駆動開発
  - SimpleModeler

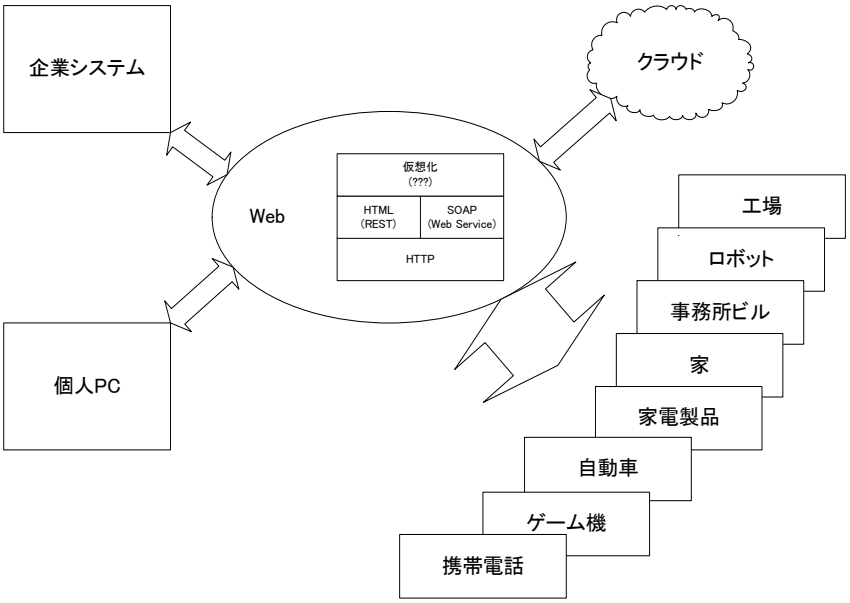
現状認識
クラウド時代

クラウドとは

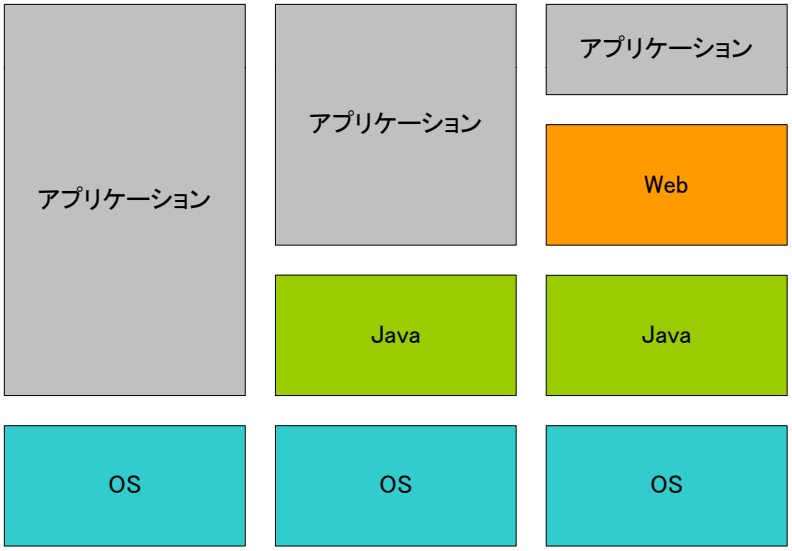


SaaS: Software as a Service
PaaS: Platform as a Service

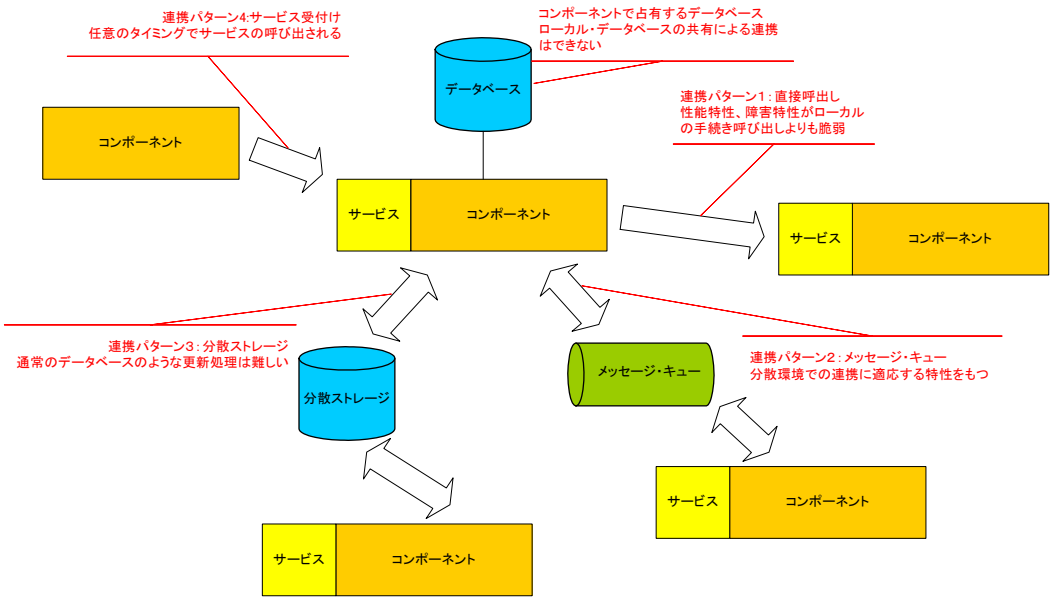
クラウド&アプライアンス・コンピューティング



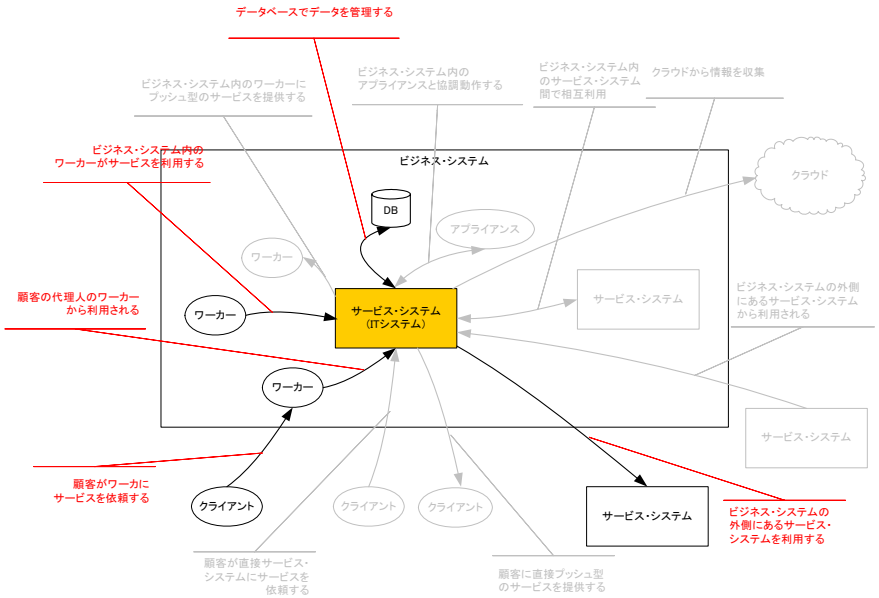
Webがプラットフォームになる



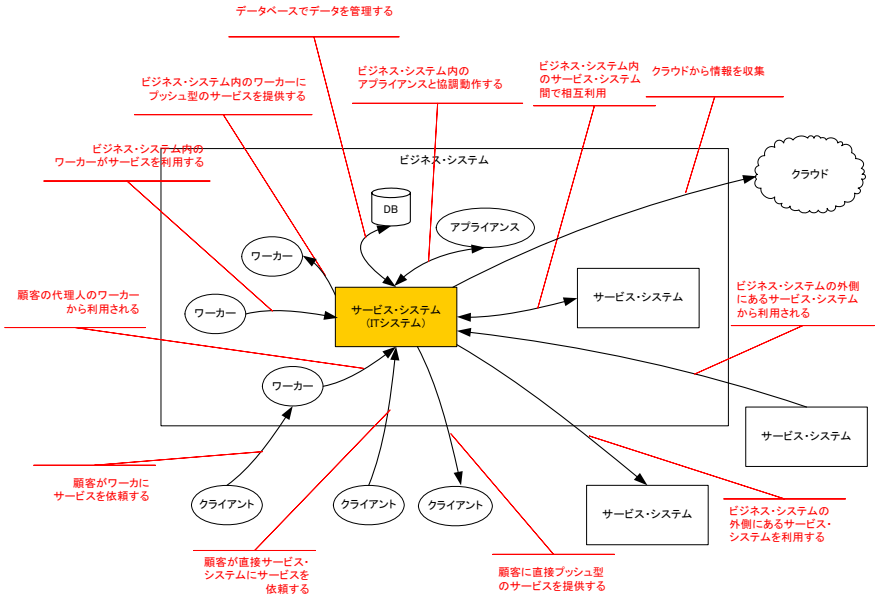
クラウド・アプリケーションの アプリケーション・アーキテクチャ



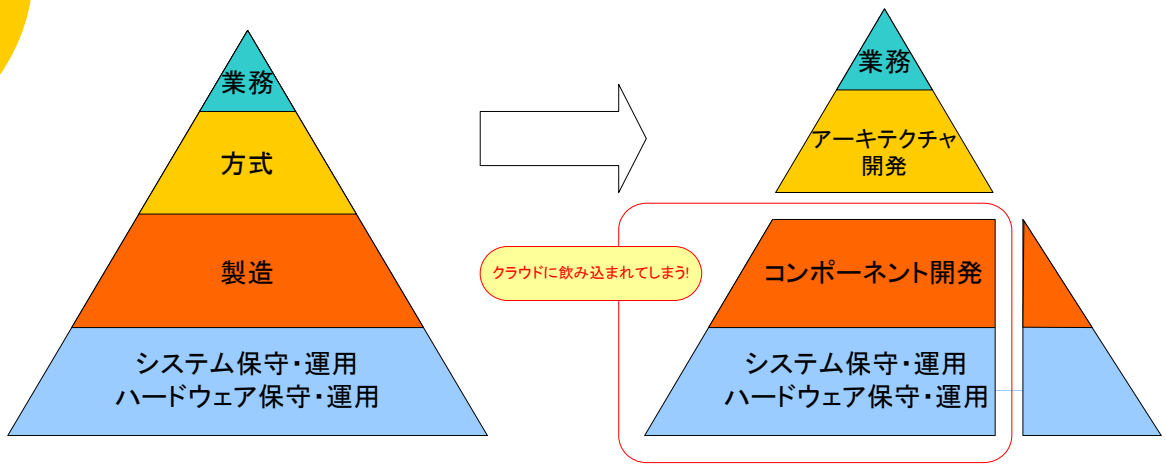
旧世代システムの相互作用



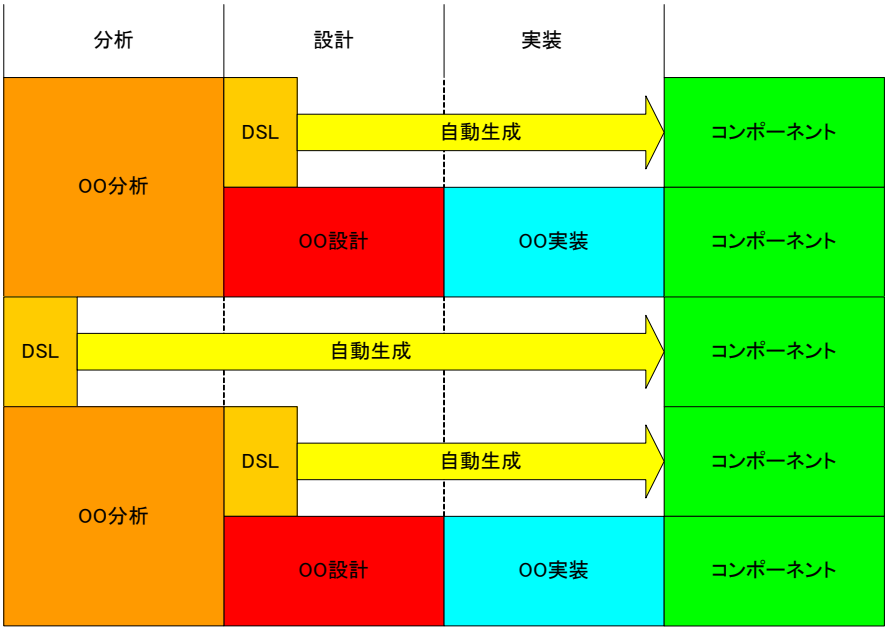
新世代システムの相互作用



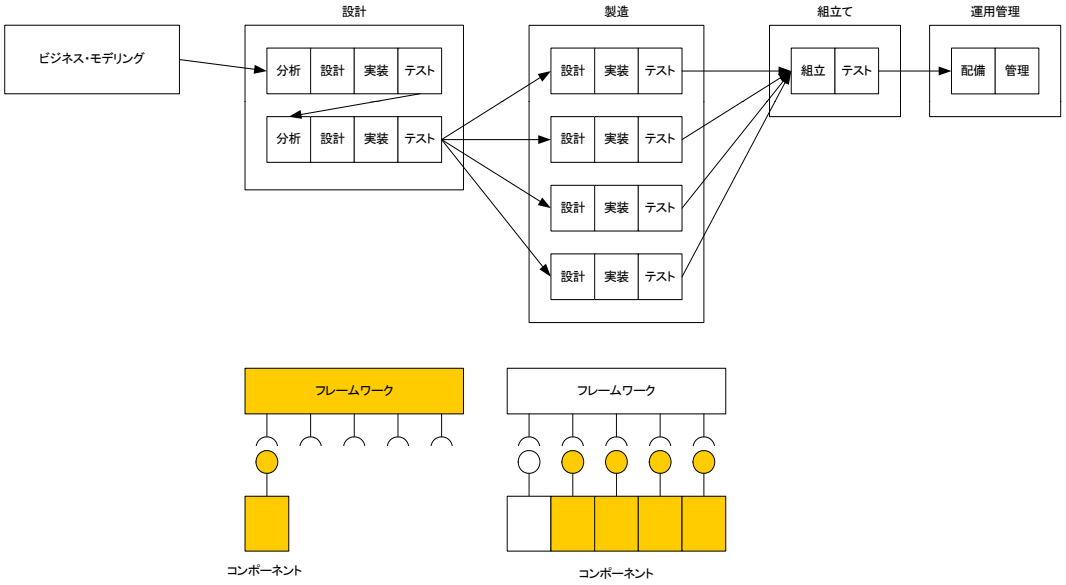
クラウド時代のソフトウェア開発



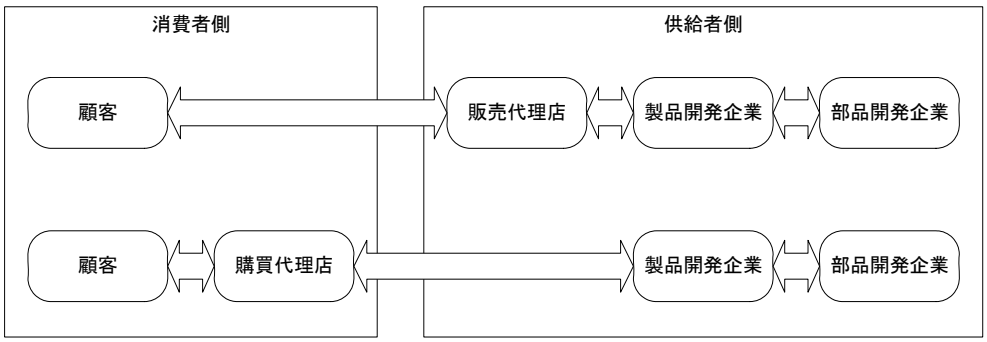
モデル駆動開発＆コンポーネント



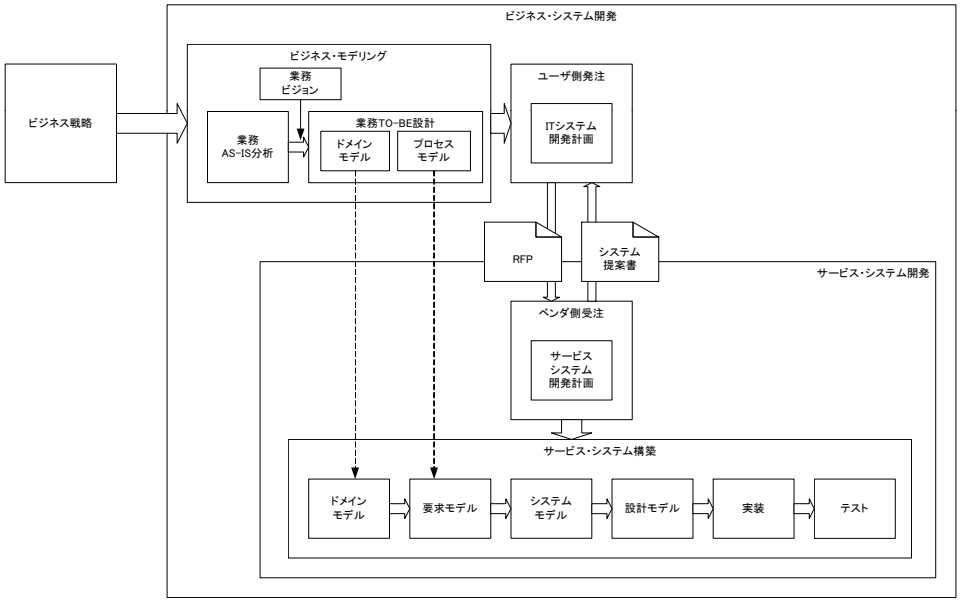
開発の流れ
Component Based Development



販売代理店から購買代理店へ



ビジネス・モデリング



クラウド時代のJavaエンジニア

- プログラミングよりモデリング
  - 問題空間(利用者視点のモデル)
    - 業務モデル、ドメイン・モデル、ユースケース・モデル
- 製造はプログラミング主導
  - 解決空間(開発者視点のモデル)モデルの必要性は薄れている
- 役割の分化
  - アーキテクチャ開発
    - アーキテクチャ・ベースラインの構築
    - Java
  - コンポーネント開発
    - Java
  - アプリケーション開発
    - 軽量言語によるマッシュアップ
    - UI開発
- ミドルウェアとしてのJavaは今後も主流
  - Javaのクラスライブラリ(業界標準API群)は大きな資産
  - JavaVM上で動作する言語をチェックしておくとよい
    - JRuby、Jython、JavaScript(Rhino)、Groovy、Scala

時代の空気 - プログラマの実感

- アジャイル
  - プログラム駆動
  - テスト駆動、振舞い駆動
  - 不必要な仕様書は作りたくない
- 軽量言語
  - スクリプト言語
  - 動的言語
  - テキスト指向
  - Web指向

プログラミング言語

- サービスのマッシュアップ
  - 軽量、テキスト指向、Web指向
  - JavaScript、Ruby、Python、Groovy、PHP、Scala(?)
- フレームワーク、サービス、コンポーネントの開発
  - 静的型付け、クラスライブラリ、標準API、ミドルウェア、分散、並行/並列、非同期
  - Java、C#、Scala

アプリケーション・アーキテクチャ

- サービス指向
  - クラウド環境
  - ビジネス・モデリング
  - モデリング技術的にはコンポーネント技術の本格適用
- 分散、並行/並列、非同期
  - コンポーネントの疎結合によるアーキテクチャ
  - 過度の仮想化(性能透過性、障害透過性)は期待しない
  - アルゴリズムから自動的に並行処理を切り出し並列・分散処理できる処理系が理想
    - 関数型言語?
- 連携方式
  - Webサービス/REST
  - 分散ストレージ
  - メッセージ・キュー

クラウド時代のソフトウェア開発技術

- 業務指向
  - 問題空間中心
    - 何を実現するのか > どのように実装するのか
  - アプリケーション構築の力点がより業務側に移ってくる
  - 業務モデルの構築と業務モデルからシステム・モデルへの落とし込みがモデリングの論点
  - 業務ユースケース/ユースケース
- コラボレーション
  - 分散環境
  - 業務ユースケース/ユースケース
- モデル駆動開発
  - 並列・分散をプログラミングするのは大変困難
  - できるだけ業務に近いモデルから自動生成
  - アジャイル開発
- CBD (Component-Based Development)
  - コンポーネントを基盤とする開発の定着を阻害する要因がなくなる
  - コンポーネントの配布、課金、広報・周知
  - コンポーネントの活用を前提とした開発

SimpleModeling

SimpleModeling

- SimpleModeling HP
  - http://simplemodeling.jp
  - MindmapModelingの基盤となるメタ・モデル
- MindmapModeling HP
  - http://mindmapmodeling.jp
  - MindmapModelingの文法、サンプルなど
  - 開発中の情報
    - http://mindmapmodeling.jp/model/index2.html
    - http://mindmapmodeling.jp/sample/yorozu/domain2.html
- JavaDSL HP
  - http://javadsl.jp
  - SimpleModelingのもう一つのDSL
- SimpleModeler
  - http://code.google.com/p/simplemodeler/
  - 最近はここがメイン

SimpleModelingのテーマ

- 教育向け、小規模開発向けのモデリング手法
  - できるだけ小さく vs. 簡略化しすぎない
  - 成果物、作業手順を明確化
- クラウド時代のモデリング手法
  - 問題空間重視
    - What > How
  - コラボレーション重視
    - 分散環境
    - ユースケース技術
  - CBD (Component-Based Development)
- モデル駆動開発
  - 具体的なプロファイル
  - DSL (Domain Specific Language)
  - アジャイル開発

SimpleModelingの特徴

- モデル化対象の絞込み
  - 教育に適した範囲
  - ただし、実務に活用できないような単純化はしない
- 業務モデルとシステム・モデルの連携
  - 業務モデル、ドメイン・モデル、要求モデル
- ユースケース
  - 利用者視点
  - 業務フローで表現できないこと
  - 業務ユースケースとシステム・ユースケース
  - コラボレーションのモデル化
- 準備モデル
  - 補助線として利用できるモデル
  - マインドマップ
- Excel
  - 帳票ベースでモデリングすることで理解が深まる
    - UMLの習得だけでは、肝心なことは分からない
- モデル駆動開発
  - モデル名プレフィックス規約
  - Scala DSL
  - アジャイル開発

"中流"モデリング

- 上流・"中流"・下流
- 上流のテーマ
  - ビジネス・モデリング
  - EA (Enterprise Architecture)
  - SOA (Service Oriented Architecture)
  - アーキテクト視点
- 下流のテーマ
  - オブジェクト指向プログラミング
  - Java, C#
  - JavaEE, .NET
  - Webサービス
  - エンジニア視点
- 中流の現状
  - ビジネス・モデリングを実装に結びつける具体的な手法
  - この分野の技術の情報が少ない

UMLの長所と短所

- 長所
  - 唯一の標準オブジェクト・モデル記法である。
  - メタ・モデルが厳密に定義されている。
  - グラフィカル言語であり、概要情報の伝達にすぐれている。
- 短所
  - オブジェクト・モデル以外の記述には必ずしも適していない。
  - オブジェクト・モデルも完全に記述できるわけではない。
  - 作成効率が必ずしも高くない。
  - モデル・リポジトリの操作性がよくない。
  - 大規模開発に必ずしも適していない。
  - 自然言語情報の取り扱いが不十分。

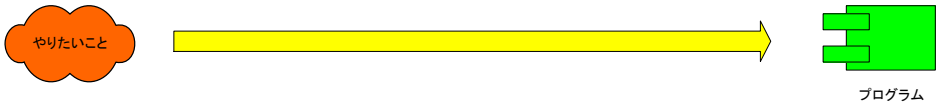
モデリング教育

- wakhokの経験
- UMLの文法やデータベース設計の知識ではモデリングはできない
- 初心者向けの本、概念的な本、個別技術を掘り下げた本はあるものの上流から下流まで網羅した教科書が見つからなかった
  - 成果物と作成方法の明確化
- 生徒(SE)が意識しているモデリング
  - 画面駆動＆データ設計(ER図)
    - 業務視点＆利用者視点の欠落
      - ビジネス･システム内での位置付け
      - 最終顧客との関係性
  - 業務フロー
    - 作業の全体像のモデル化には有効
    - 作業の網羅性、例外処理の記述性に問題

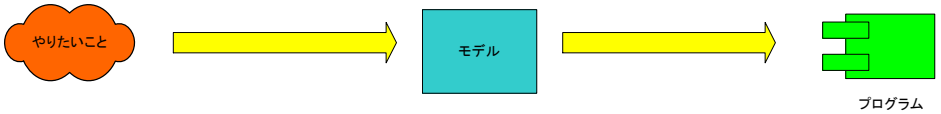
モデリング教育

- モデリングのコツをいかに伝授するのか
- UMLの習得(のみ)を目指すのは効果が薄い
  - UMLではモデルの詳細を記述することが難しい
  - 実際の開発に必要なUMLの機能は非常に限られているので、UMLを網羅的に学ぶのは非効率
- ゼミ･OJTが望ましい
  - 講義形式の授業は効果が薄い
  - 課題に対して具体的に成果物を作成し、レビューによるフィードバックが効果的
  - 教科書を覚えるのではなく、講師のスキルを盗む

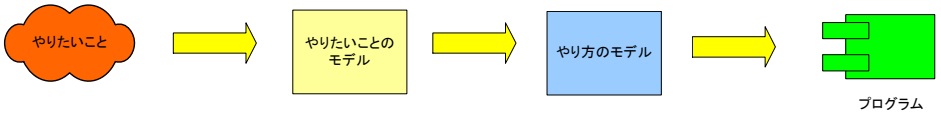
モデリングの意味



(1) “やりたいこと”とプログラムの間の距離は長い

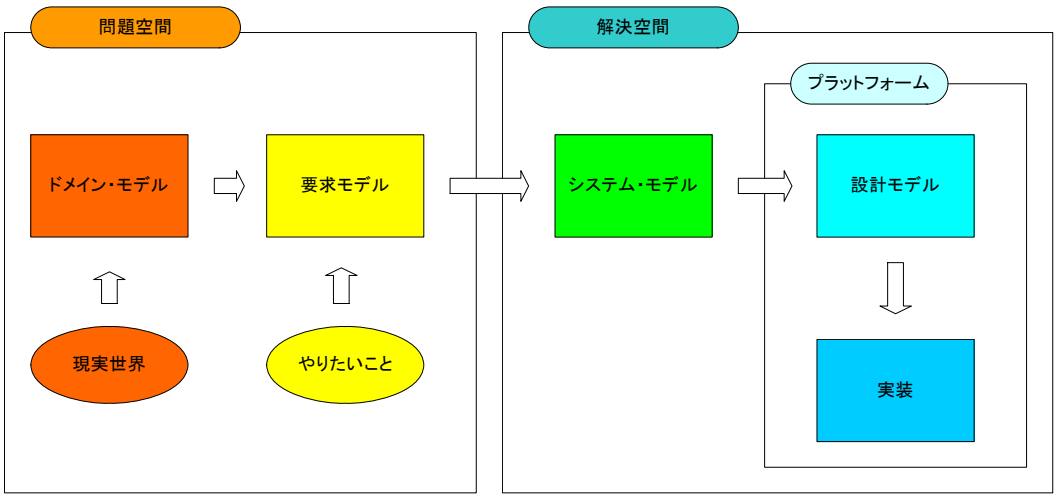


(2) “やりたいこと”とプログラムの間をモデルで中継

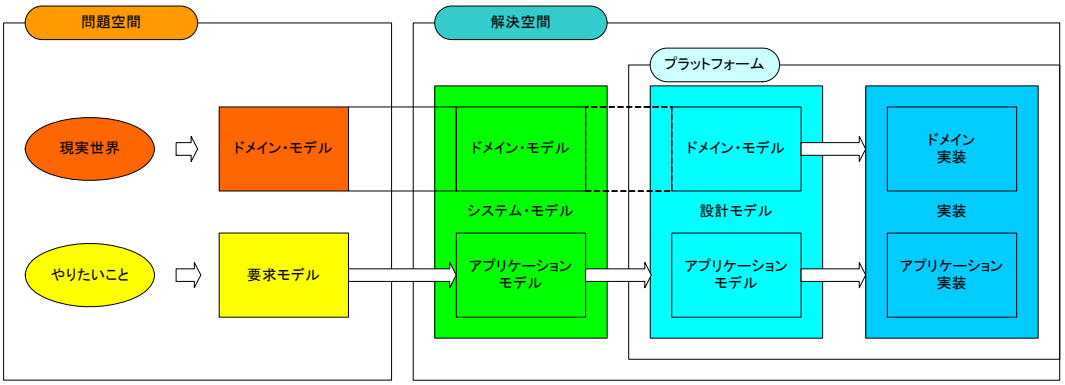


(3) “やりたいこと”のモデルとやり方のモデル

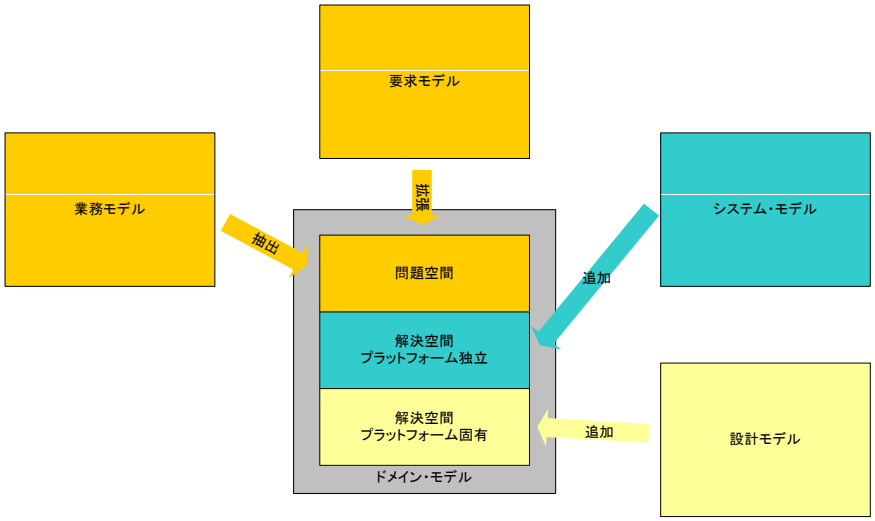
SimpleModeling
モデル変換/作業の流れ



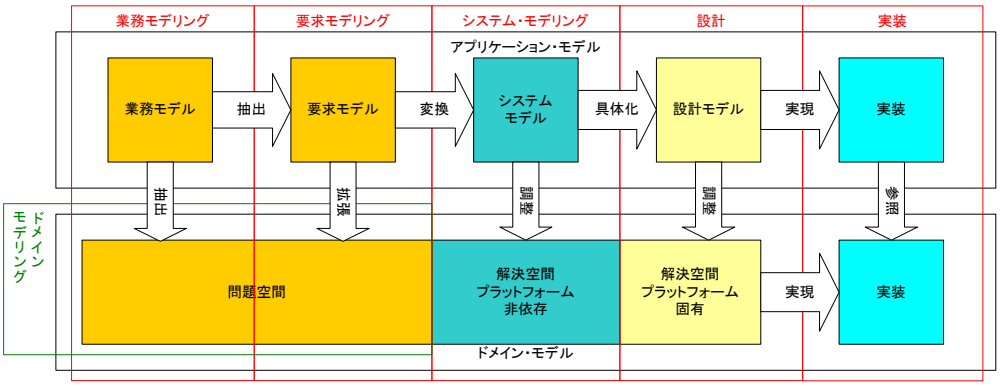
SimpleModeling
モデル変換/モデルの観点から



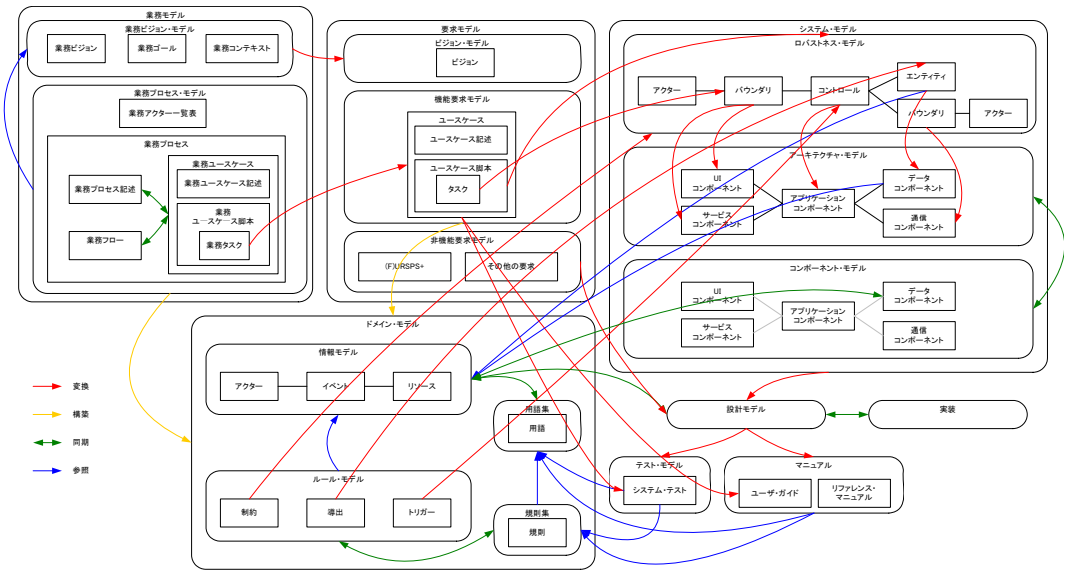
SimpleModeling
ドメイン・モデルをハブとした連携



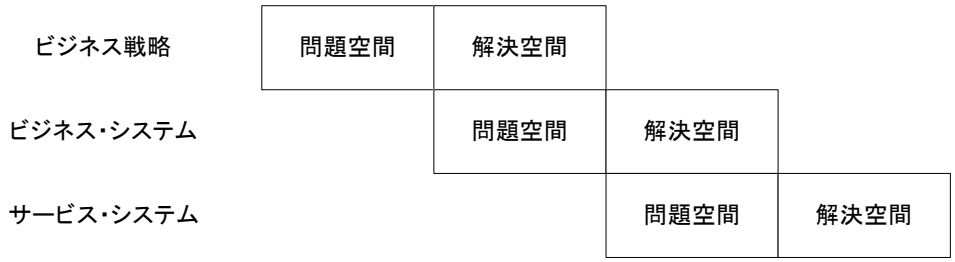
SimpleModeling
モデル変換の流れ



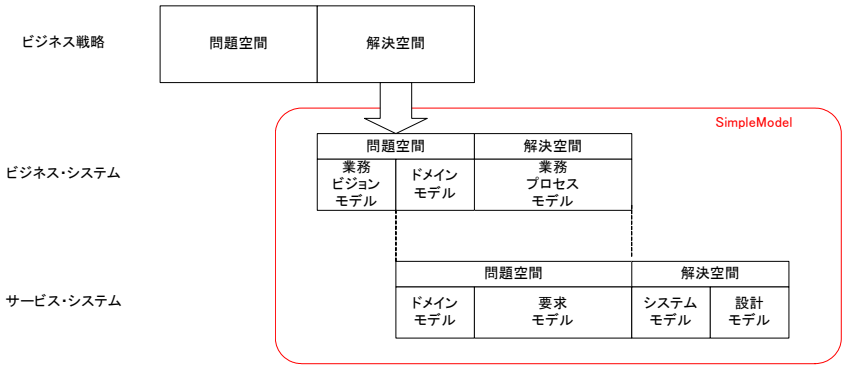
モデル体系



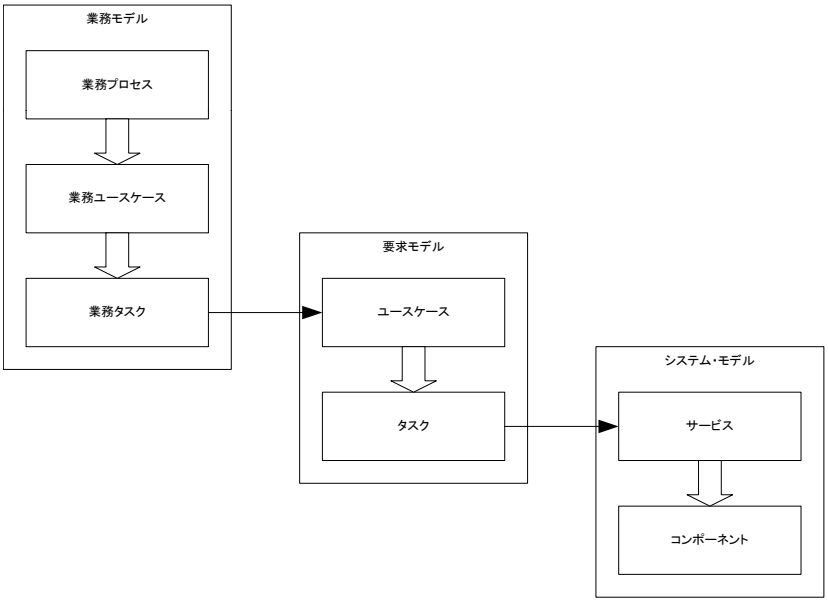
企業システム・モデリングの3階層構造



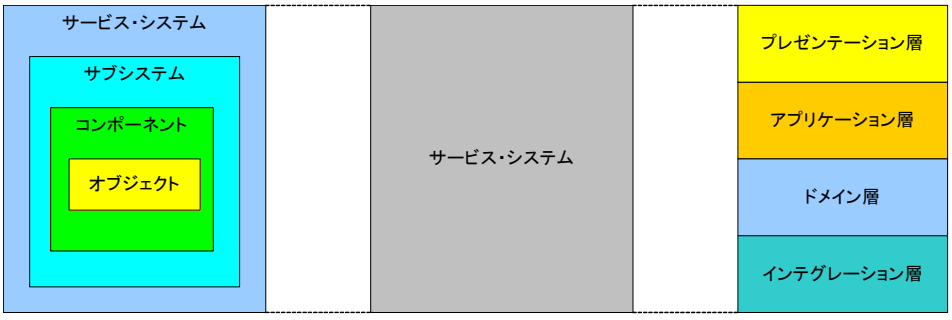
SimpleModelの対象範囲



SimpleModeling
軸となるモデル連携



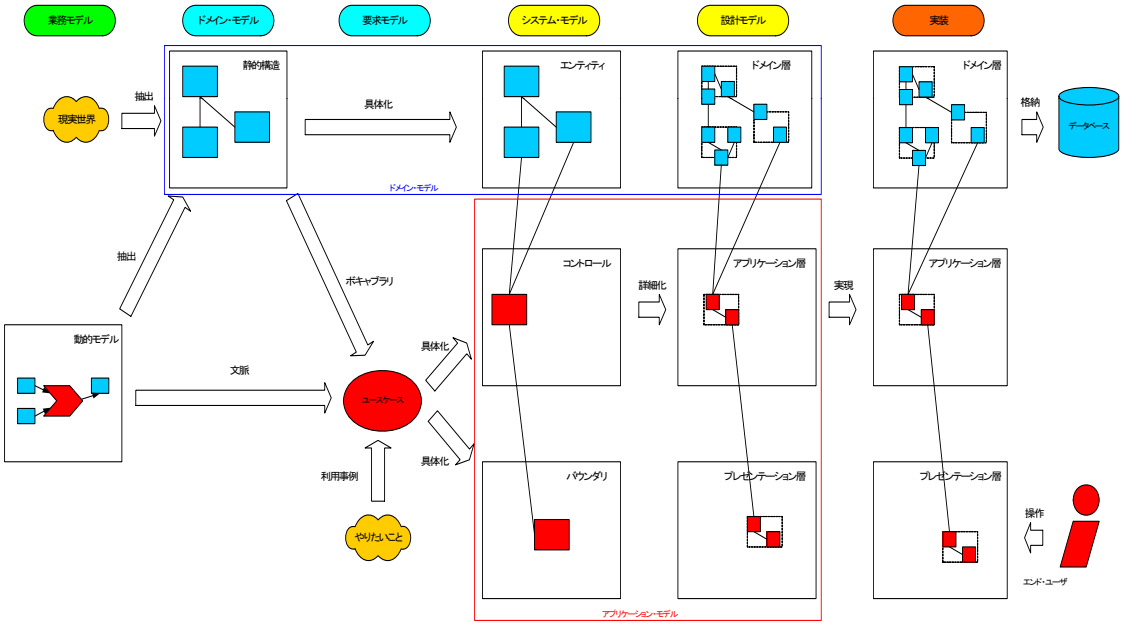
サービス・システムの構成



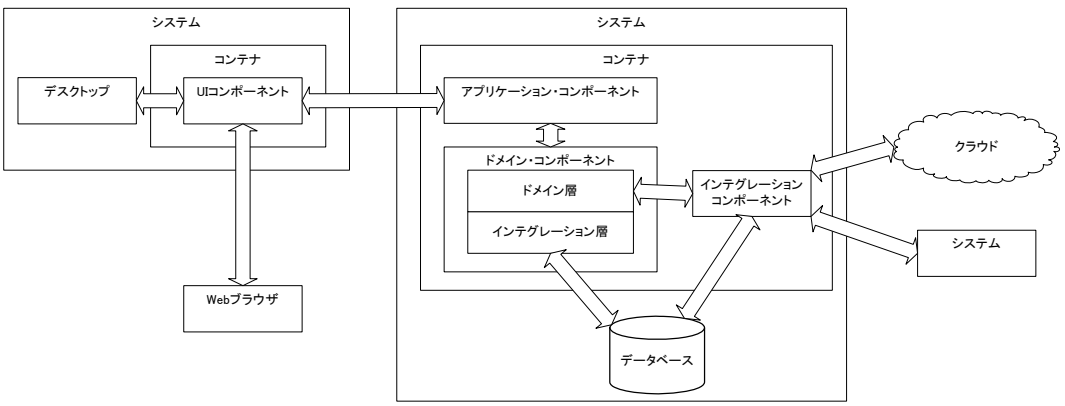
論理モデル視点

アーキテクチャ層ビュー

モデルとアーキテクチャ



システム・アーキテクチャ



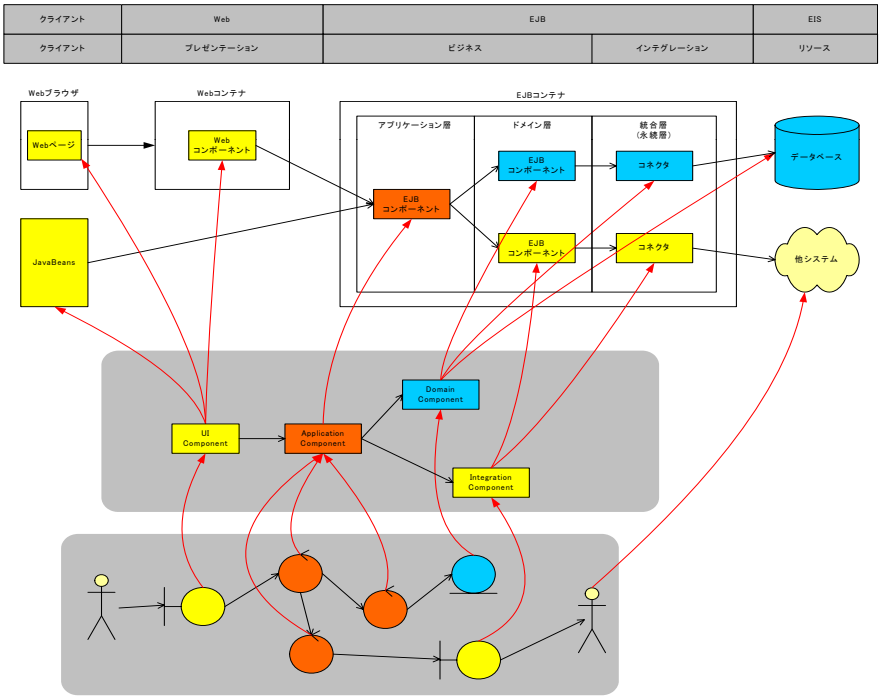
解決空間モデル

- システム・モデル
  - PIM(Platform Independent Model)
  - プラットフォーム独立の抽象度の高いモデル
- 設計モデル
  - PSM(Platform Specific Model)
  - Java/JEEによる実装のためのモデル

SimpleModelingのPIMモデル

- PIM(Platform Independent Model)
- ドメイン・モデル
  - 情報モデル
  - ルール・モデル
- システム・モデル
  - システム・アーキテクチャ・モデル
  - コンポーネント・モデル
  - モジュール・モデル

コンポーネントとJavaEE



ドメイン・モデル

- ドメイン・コンポーネントとして実現
  - 外部仕様は通常のコンポーネント
- エンティティやルールはドメイン・コンポーネント内のオブジェクトとして実現
- 公開方法
  - オペレーション経由で間接的に公開
  - オブジェクトを直接公開
- 永続化の実現場所
  - ドメイン・コンポーネント内で実現
  - インテグレーション・コンポーネントを使用
- 永続化の実現方法
  - プログラミング
  - O/Rマッピング

システム・モデル

- システム・アーキテクチャ・モデル
  - コンポーネントを組み立ててシステムを構築する。
- コンポーネント・モデル
  - UIコンポーネント
    - User Interfaceを実現
  - サービス・コンポーネント
    - 他システムに公開するサービスを実現
  - アプリケーション・コンポーネント
    - アプリケーション・ロジックを実現
  - ドメイン・コンポーネント
    - ドメイン・モデルを実現(情報モデル、ルール・モデル)
  - インテグレーション・コンポーネント
    - システム・リソース(データベースなど)へのアクセス
    - 他システム、サービスとの通信を実現
- モジュール・モデル
  - 配備の単位、流通の単位、開発の単位でコンポーネントを束ねる

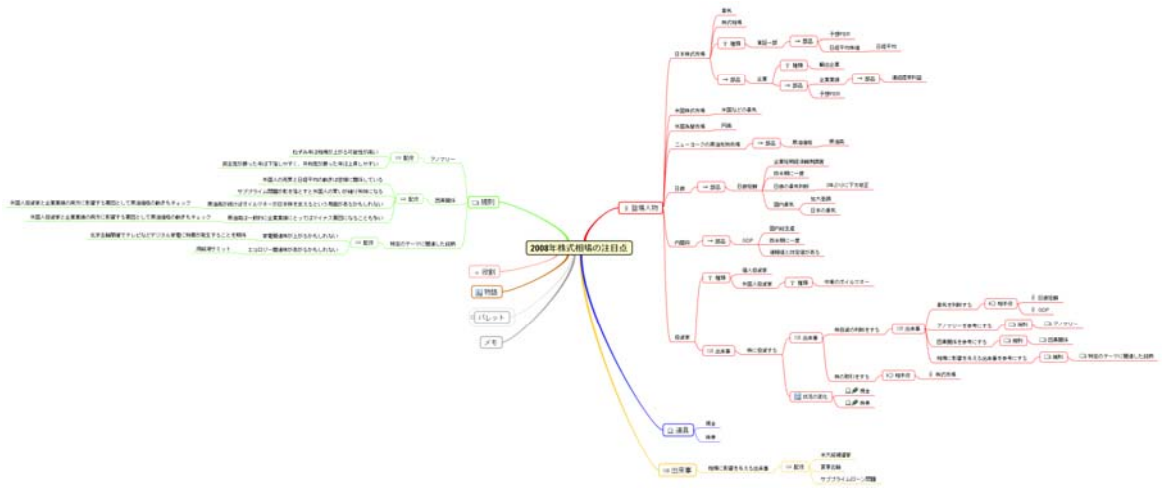
サービスとコンポーネント

- サービス
  - 再利用可能な部品
  - 利用者の目標を解決するための機能とインタフェース
  - プログラム呼出し以外の方法での利用が主
    - UI、RPC(SOAP、IIOPなど)、REST
    - 遠隔呼出しに適した粒度と信頼性
      - 大きな粒度
      - 低信頼性を想定
    - RESTが重要な理由の一つは、UIとAPIの両方を同時に提供しているから
- コンポーネント
  - 再利用可能な部品
  - 提供者が提供できる機能とインタフェース
  - プログラム呼出しでの利用が主
    - 静的な結合(少なくてもプログラム起動時)
    - 遠隔呼出しには必ずしも適さない粒度と信頼性
      - 小さな粒度
      - 高信頼性を想定

マインドマップ・モデリング

- マインドマップを記述言語とするモデリング手法
  - SimpleModelingのマインドマップ記法
- 教育用
  - オブジェクト・モデリングのバックボーン
    - ドメイン・モデルとユースケース・モデル
  - ゼミで利用
  - 実開発でも準備モデルとして利用できると考えている
- モデリングのコツをどのようにして会得するのか
  - 直感的に分かりやすい
    - 演劇のメタファ
    - 構造と物語
  - イベントの発生とドメイン・オブジェクトの状態遷移

マインドマップによるモデル記述

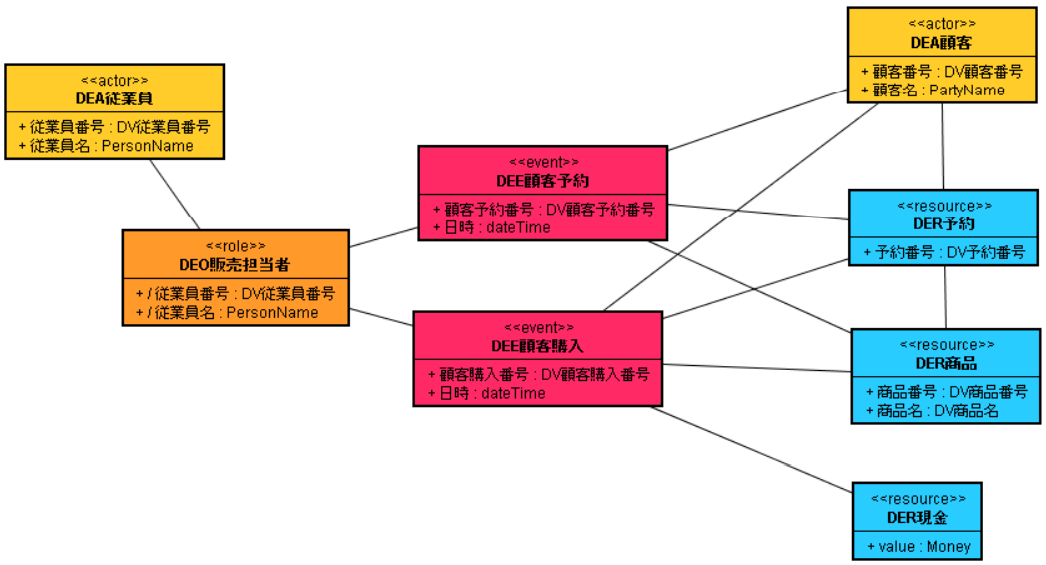


モデル・サンプル

ビジョン宣言

業務ビジョン宣言	2008-02-20/浅海	
概要		備考
このビジョンの目的	よろず商会販売プロセス	
業務プロセスの概要	美術品販売と販売済み商品の委託販売を行う。	
参照する情報		
ユーザ		備考
マーケットの状況	固定顧客の高齢化が進む。若年層の取り込みが課	
ユーザはどんな人か	独身で都会で一人暮らし。または子供なしの夫婦。	
ユーザの利用環境	都会のマンションでインテリアとして利用。	
ユーザがやって欲しいこと	美術品をインテリアの小物として気軽に利用したい。	
他の解決手段	雑貨小物をインテリアとして利用。	
ポジション宣言		備考
ターゲットの顧客(For)	美術品を購入する顧客	
顧客がやって欲しいこと(Who)	美術品をインテリアとして使用して季節ごとに取り替えたい	
製品名(The)	よろず商会販売プロセス	
カテゴリ(Is a)	美術品販売プロセス	
使うメリット(That)	販売後の美術品を、インテリアとして使用したまま委託販売できる	
代わりの方法(Unlike)	商品を販売	
優位な相違点(Differenciation)	美術品を販売委託中もインテリアとして利用できるので、買い替えを行いやすい	
効用とフィーチャ(特徴)		備考
顧客にとっての効用	効用を実現するフィーチャ(特徴)	
美術品購入	美術品販売	
使用しなくなった美術品を売る	商品を預ける美術品委託販売	
使用中の美術品を売る	商品を所有したままの美術品委託販売	
業務プロセス構築の前提条件		備考
仮定していること		
依存しているもの		
コスト		
業務プロセスに対するその他の要求		備考
標準規約		
文書に対する要求		備考
ユーザ・ガイド		
リファレンス・マニュアル		
インストール・ガイド		
運用マニュアル		

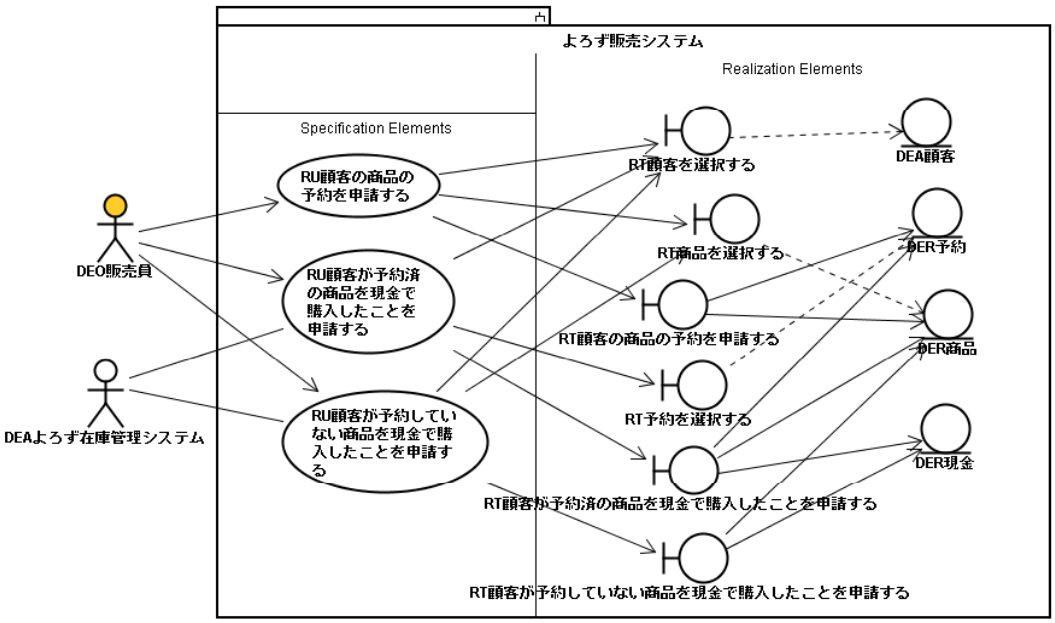
ドメイン・エンティティ図



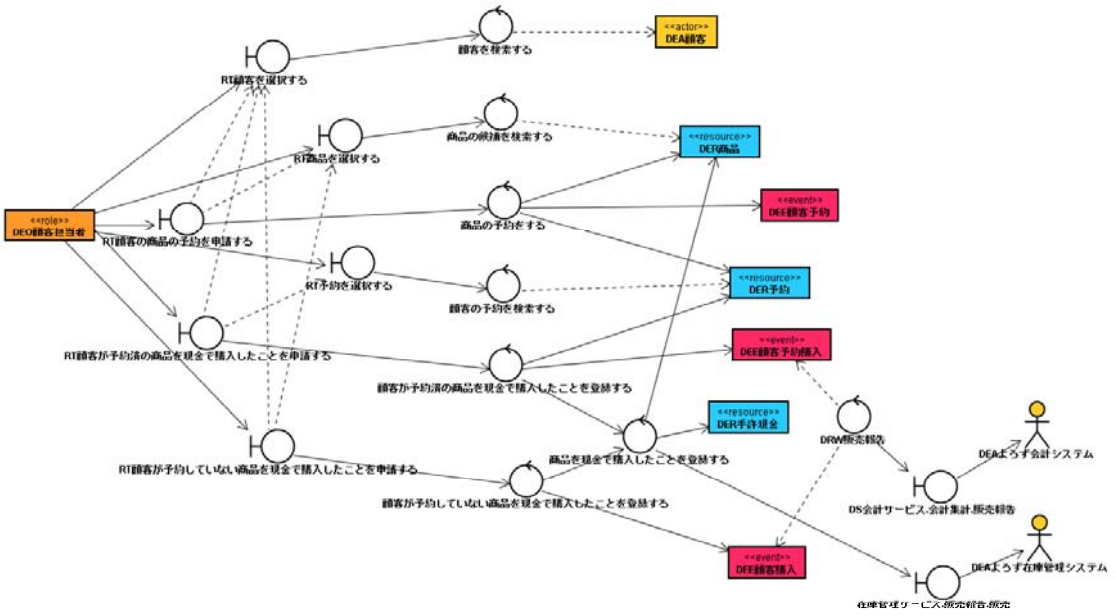
ドメイン・オブジェクト記述

ドメイン・オブジェクト記述							2008-06-01/浅海
ID							
名前		DEA顧客					
用語名	日本語名	顧客					
	英語名						
実現名	プログラム						
	データベース						
所属	コンポーネント	よろず販売					
	パッケージ						
関係	基底オブジェク						
	派生オブジェク	DEA法人顧客, DEA個人顧客					
特性	種別	actor					
	種別	client					
	区分						
	注記						
概要	見出し						
	概要						
説明		よろず商会と取引する顧客。よろず商会から美術品を購入、よろず商会に美術品を販売する。					
備考							
属性							
名前	型	コンポーネント	パッケージ	多重度	注記	説明	備考
顧客番号	DV顧客番号			1	ID		
顧客名	PartyName			1			
関連							
名前	型	コンポーネント	パッケージ	多重度	注記	説明	備考
操作							
名前	入力	出力			注記	説明	備考
状態							
状態	イベント	ガード	遷移先			説明	備考
サービス							
サービス	ポート	操作	入力	出力		説明	備考

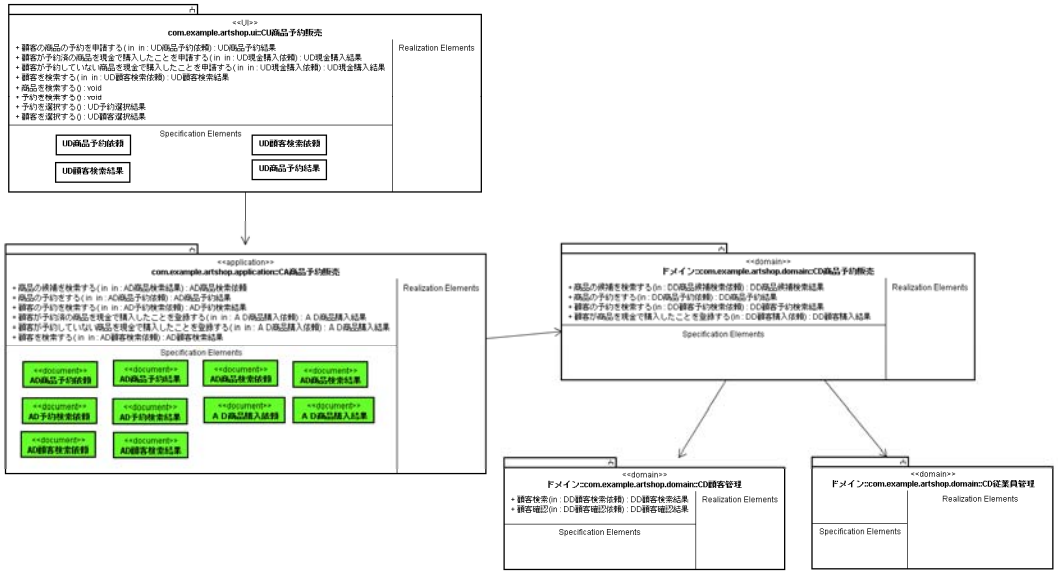
ユースケース図



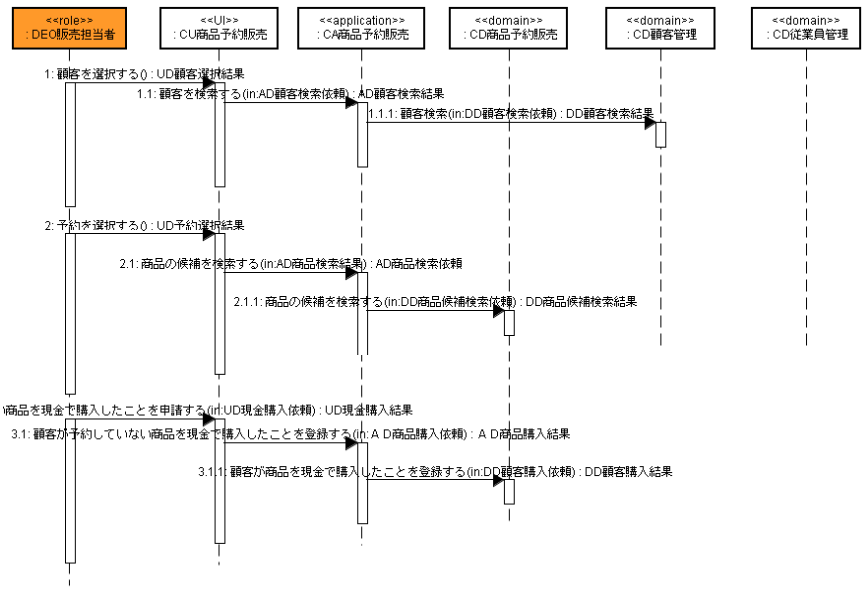
ロバストネス図



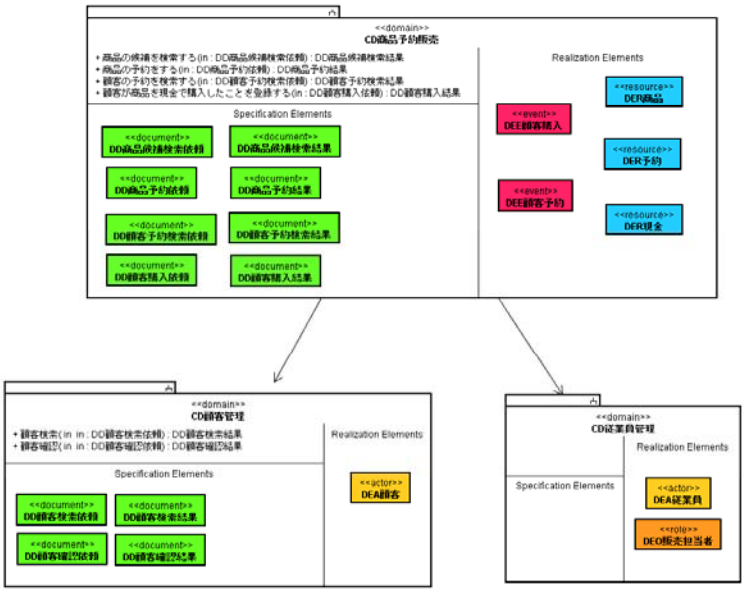
システム・アーキテクチャ図



ユースケース・シーケンス図



ドメイン・コンポーネント図



Javaプログラミング

Javaプログラミングでの考え方

- PIMモデルをそのまま入力とする
  - コンポーネントの仕様が定まっていれば十分
  - Javaはオブジェクト指向言語として十分な機能を持っているので、いわゆる”詳細設計”は不要
- 設計が必要な場合
  - GUIの画面構成やデータベースの物理モデルなど、Javaプログラミング以外のもの

Javaプログラミング

- ドメイン・モデル
  - ドメイン・コンポーネントとして実現
- コンポーネント・モデル
  - コンポーネント
  - モジュール
- システム・アーキテクチャ・モデル
  - コンポーネントの組立て

Javaプログラミング
システム・アーキテクチャ・モデル

- コンポーネントの組立てとして実現
- JEE
  - 配備ディスクリプタ
- DIコンテナ
  - 定義ファイル、命名規約

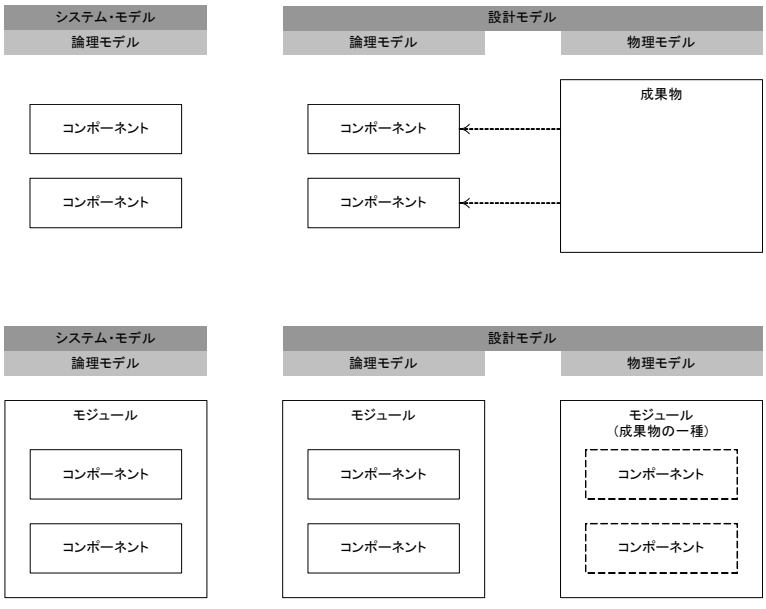
Javaプログラミング
コンポーネント？モジュール？

- JavaBeans
  - JAR (Java Archive)
    - Java classファイル
    - META-INF/MANIFEST.MF
- Enterprise JavaBeans
  - EJB-JAR
    - Java classファイル
    - META-INF/ejb-jar.xml
  - WAR (Web Archive)
    - Java classファイル, HTML, JSP
    - WEB-INF/web.xml
  - RAR (Resource Archive)
    - Java classファイル
    - META-INF/ra.xml
  - Client-JAR
    - Java classファイル
    - META-INF/client-jar.xml

Javaプログラミング
コンポーネントとモジュール

- コンポーネントとモジュールの考え方を整理しなければならない
  - コンポーネントの一般的な定義:再利用可能なソフトウェア部品
- UMLでは…
  - UML 1.x:コンポーネントは配備の単位(物理的なモデル要素)
  - UML 2.x:コンポーネントは配備の単位(物理的なモデル要素)かつ再利用可能なソフトウェア部品(論理的なモデル要素)
  - モジュールというモデル要素はない
- 今後は以下のように整理されてくると思う。
  - コンポーネント:再利用可能なソフトウェア部品
  - モジュール:配備の単位

コンポーネントとモジュール



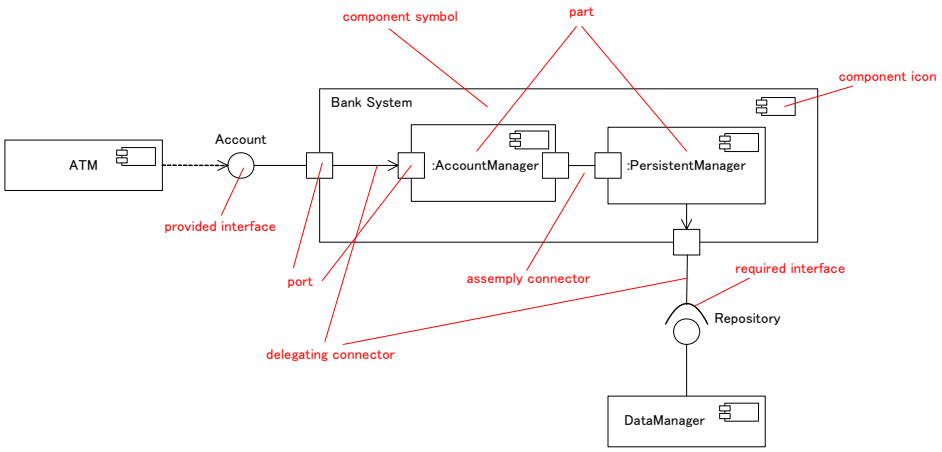
Javaプログラミング モジュール

- JavaBeans
  - JAR (Java Archive)
- Enterprise JavaBeans
  - EJB-JAR
  - WAR
  - RAR
  - Client-JAR
- Maven2
  - POM(Project Object Model)
- Java7
  - JAM(Java Application Modules)
    - JSR 277:Java Module System
    - JSR 294:SuperPackage

Javaプログラミング コンポーネント

- 「論理的なソフトウェア部品」
  - 物理的な側面の表現はモジュールに移動
- コンポーネントを集めて配備の単位であるモジュールを作成する

UMLコンポーネント図



Javaプログラミング
コンポーネントの実現

- パート(part)
  - Javaクラス
- 提供インタフェース(provided interface)
  - Javaインタフェース
  - UI(UIコンポーネントの場合)
  - Web(REST, SOAP)
- 必要インタフェース(required interface)
  - Javaインタフェース
  - Web(REST, SOAP)
- ポート(port)
  - メソッドの集まり
  - UI部品(画面など)の集まり(UIコンポーネントの場合)
  - WSDL/Port
- 委譲コネクタ(delegating connector)
  - Javaインタフェース&メソッド
  - ツールによる自動生成
- 組立てコネクタ(assembly connector)
  - インスタンス変数
  - プログラム(糊コード)またはDIコンテナで設定

Javaプログラミング
コンポーネントの種類(1)

- UIコンポーネント
  - User Interfaceを実現
  - HTML, JSP, JSF
  - Swing
- サービス・コンポーネント
  - 他システムに公開するサービスを実現
  - JAX-WS
  - RMI
- アプリケーション・コンポーネント
  - アプリケーション・ロジックを実現
  - Javaオブジェクト

Javaプログラミング
コンポーネントの種類(2)

- ドメイン・コンポーネント
  - ドメイン・モデルを実現(情報モデル、ルール・モデル)
  - JDBC
  - JPA, Hibernate
- インテグレーション・コンポーネント
  - システム・リソース(データベースなど)へのアクセス
  - 他システム、サービスとの通信を実現
  - JDBC
  - JPA, Hibernate
  - JMS
  - JCA
  - JAX-WS
  - RMI

Javaプログラミング
ドメイン・モデル

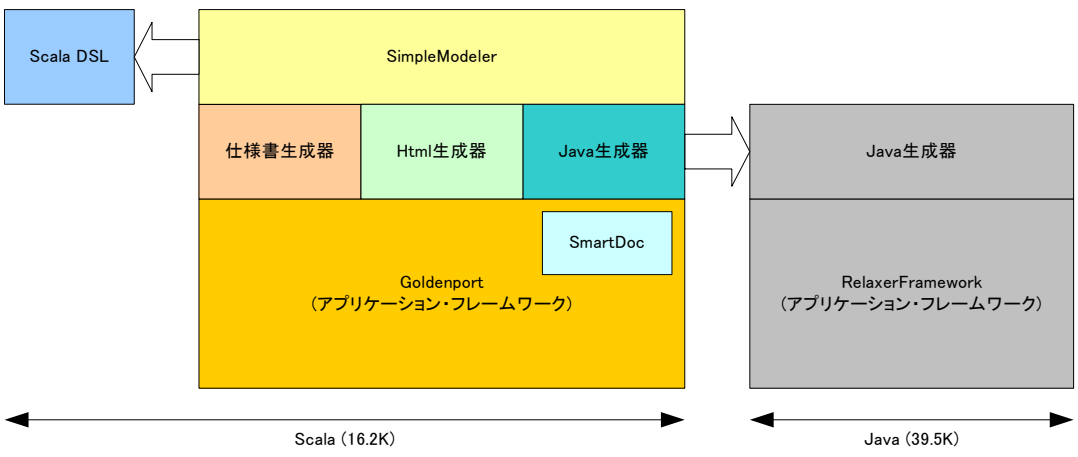
- ドメイン・コンポーネントとして実現
  - 外部仕様は通常のコンポーネント
- エンティティやルールはドメイン・コンポーネント内のJavaオブジェクトとして実現
- 公開方法
  - メソッド経由で間接的に公開
  - Javaオブジェクトを直接公開
- 永続化の実現場所
  - ドメイン・コンポーネント内で実現
  - インテグレーション・コンポーネントを使用
- 永続化の実現方法
  - プログラミング
    - JDBC
  - O/Rマッピング
    - JPA, Hibernate

SimpleModeler

SimpleModelerとは

- SimpleModel(SimpleModelingのメタ・モデル)のモデル・コンパイラ
- Scalaをメタ言語としたDSL(Domain Specific Language)
  - ScalaはJava VM上で動作するOO+関数型言語

構成



SimpleModelerで実現したいこと

- ユースケース・モデルとドメイン・モデルの連携
- テキストによるモデル記述
  - テキストベースのDSL
- プログラムの自動生成
  - ドメイン・モデルを中心として
- 仕様書の自動生成
  - 仕様書生成のための仕掛け
- 仕様検証
- 教育用途
- コンサルティング
- テキスト・エディタ＋オープン・ソースで手軽に

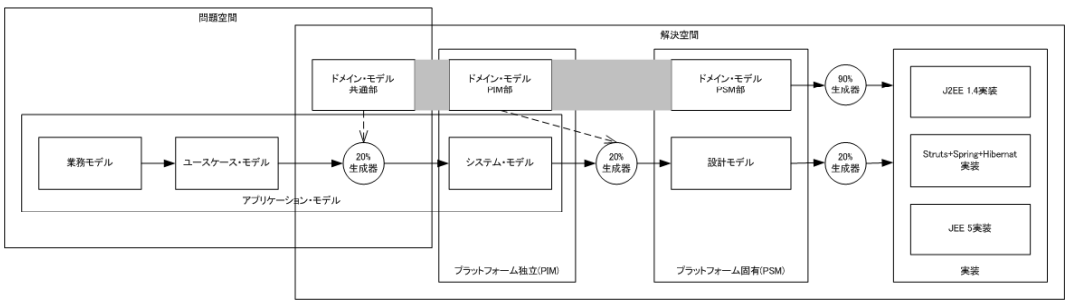
DSL駆動開発の論点 SimpleModelerの選択

- 自動生成できる範囲を明確にする。
- 適用対象を絞り込む。
- 開発プロセスを定義する。
- 主力の記述言語にUMLを使用しない。
- 主力の記述言語にグラフィカル言語を使用しない。
- メタモデルの定義にMOFを使用しない。
- 記述言語はツールからの操作性を重視する。
- 自然言語情報を定型的に取り扱える。

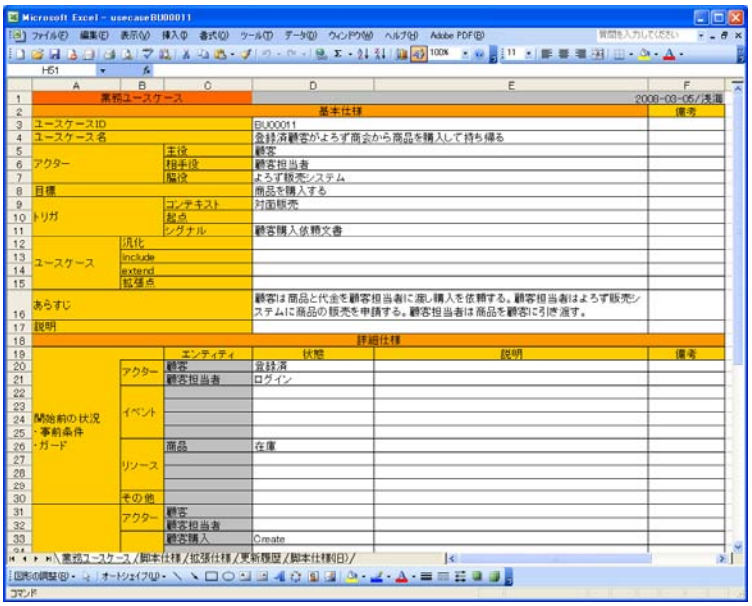
DSL実現の選択肢

- UML
  - グラフィカル言語は編集が煩雑
  - 帳票形式の情報をうまく扱えない
- Excel
  - 物理構造が表形式の集まりに限定される
- マインドマップ
  - 精密な記述ができない
- 動的型付け言語
  - コンパイル･エラーによるエラー検出の範囲が小さい
  - IDEによる入力補完の精度が低い
- Java
  - Javaを採用したJavaDSLを開発していたが中断
- Scala
  - Scalaを採用したDSLを開発中

ツールによる自動生成(構想)



Excelによるモデル記述



	A	B	C	D	E	F
1	業務ユースケース					2008-03-05/浅海
2				基本仕様		備考
3	ユースケースID			BU00011		
4	ユースケース名			登録済顧客がよろず商会から商品を購入して持ち帰る		
5	アクター		主役	顧客		
6			相手役	顧客担当者		
7			脇役	よろず販売システム		
8	目標			商品を購入する		
9	トリガ		コンテキスト	対面販売		
10			起点			
11			シグナル	顧客購入依頼文書		
12	ユースケース	汎化				
13		include				
14		extend				
15		拡張点				
16	あらすじ			顧客は商品と代金を顧客担当者に渡し購入を依頼する。顧客担当者はよろず販売システムに商品の販売を申請する。顧客担当者は商品を顧客に引き渡す。		
17	説明					
18				詳細仕様		
19			エンティティ	状態	説明	備考
20	開始前の状況 ・事前条件 ・ガード	アクター	顧客	登録済		
21			顧客担当者	ログイン		
22		イベント				
23						
24						
25						
26		リソース	商品	在庫		
27						
28						
29						
30		その他				
31		アクター	顧客			
32			顧客担当者			
33			顧客購入	Create		

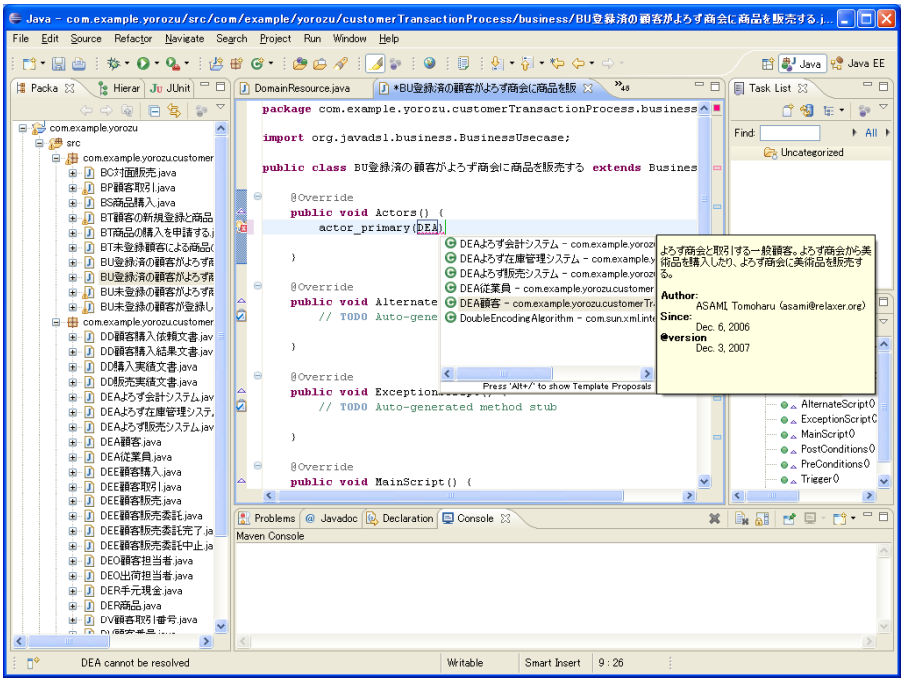
JavaDSL

- Excelベースでのモデル作成の限界
- JavaベースのDSL
- ツール(Relaxer)でプログラムの自動生成
- SimpleModel記述のためのDSLの開発
  - UML、Excel、マインドマップはいずれも正確なモデル記述が難しい
- DSLから仕様書やプログラムの自動生成
  - 作成したモデルの”見える化”
  - プロトタイプ・システム開発

JavaDSL
ドメイン・アクター「DEA顧客」

```
/**
 * <p summary="true"><span headline="true">よろず商会と取引する一般顧客</span>。よろず商会から美術品を購入したり、よろず
       商会に美術品を販売する。</p>
 */
public class DEA顧客 extends DomainActor {
   @DomainProperty(DomainPropertyType.ID)
   public DV顧客番号 顧客番号;

   /**
    * <p summary="true"><span headline="true">顧客名</span>。</p>
    *
    * [顧客]は個人の場合と法人の場合があるのでデータ型は[PartyName]となっている。
    */
   @DomainProperty
   public PartyName 名前;

   /**
    * <p summary="true">顧客の住所。</p>
    */
   @DomainProperty
   public PartyAddress 住所;

   @Override
   public void Information() {
      title("顧客");
   }
}
```

Eclipseによるモデル作成



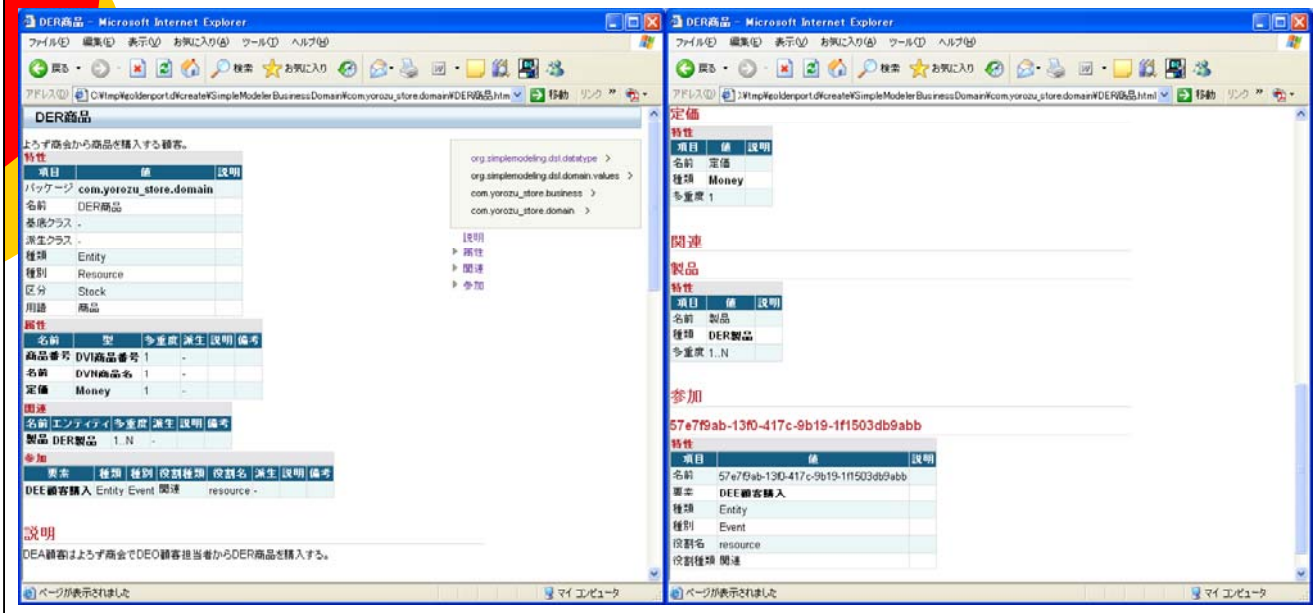
“モデグラム”

- モデル＆プログラム
- 浅海の造語
- アジャイルとモデリングを組み合わせるためには、モデリングがプログラミングにならなければならない。
- UMLでプログラミングするのは実用的ではない。
- オブジェクト・モデルをテキスト・ベースのDSLで記述する

SimpleModel DSL
ドメイン・リソース「DER商品」

```
[packageとimportは省略]

case class DER商品 extends DomainResource {
  term = "商品"
  caption = "よろず商会が販売する商品"
  brief = <t>商品は複数の製品から構成されている。</t>
  description = <text>よろず商会が販売する商品。顧客は顧客担当者から商品を購
    入する。</text>

  resource_type is Stock because "商品は在庫として管理する。"
  resource_unit_type is Individual because "商品は個別に管理する。"

  id("商品番号", DVI商品番号())
  attribute("名前", DVN商品名())
  attribute("定価", Money)

  "製品" is_one_more DER製品()
}
```

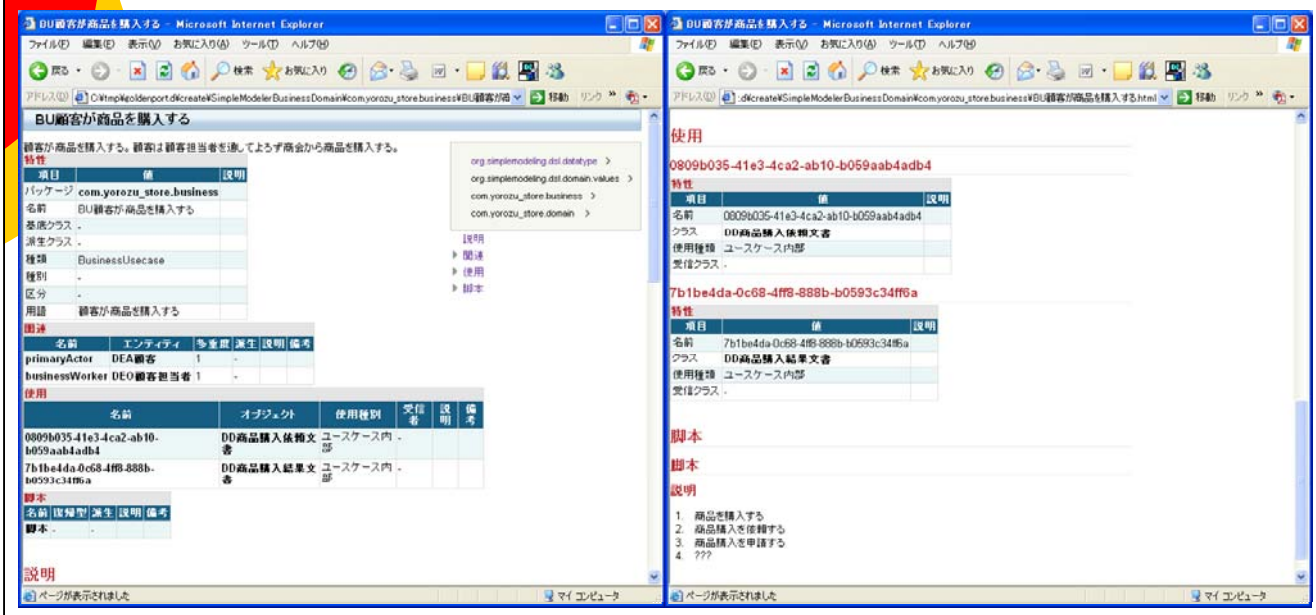
ドメイン・リソース「DER商品」



SimpleModel DSL
業務ユースケース「BU顧客が商品を購入する」

```
case class BU顧客が商品を購入する extends BusinessUsecase {
  caption = <t>顧客が商品を購入する。</t>
  brief = <t>顧客は顧客担当者を通してよろず商会から商品を購入する。</t>
  description =
    <text>
  <p>顧客はよろず商会で顧客担当者から商品を購入する。</p>
  <p>顧客は顧客番号によって識別される。</p>
  <p>顧客は名前と住所をデータとして管理される。</p>
    </text>

  actor is_a DEA顧客()
  worker is_a DEO顧客担当者()

  basic_flow {
    task("商品を購入する") {
      step_actor_worker("商品購入を依頼する", DD商品購入依頼文書(), DD商品購入結果文書()) {
        step_worker_system("商品購入を申請する", DD商品購入依頼文書(), DD商品購入結果文書()) {
          step_system("顧客購入を実行する") {
            event_issue(DEE顧客購入()) {
              resource_update(DER商品())
            }
          }
        } mark_is "buy"
      }
    }
  }
[省略]
}
```

SimpleModel DSL
業務ユースケース「BU顧客が商品を購入する」



まとめ

- モデル駆動開発が次世代ソフトウェア開発技術の焦点
  - MDAは優秀な教科書として考えているが、直接実務への適用は困難ではないかというのが私見
    - UMLでモデル作成するのは辛い(施工図ならOK)
    - 純粋なUMLは汎用的すぎる(応用に合わせたチューニングが必要)
    - 動的モデルがプログラミングとインピーダンス・ミスマッチ
- SimpleModeling
  - モデル駆動開発をターゲットにしたモデリング手法
- SimpleModeler
  - SimpleModeling用モデル・コンパイラ
  - Scala DSLによるモデル記述
  - SimpleModeler仕様書、プログラム自動生成